ストップウォッチオートマトンによるプリエンプティブスケジューリングの仕様記述と有界モデル検査 (理論計算機科学の深化 : 新たな計算世界観を求めて)

AUTHOR(S):

瀧内, 新悟; 山根, 智

数理解析研究所講究録
第1599巻 2008年 1-8

2007年度冬のLAシンポジウム [7]

ストップウォッチオートマトンによるプリエンプティブスケジューリングの仕様記述と有界モデル検査

瀧内 新悟, 山根 智
金沢大学大学院自然科学研究科
takinai @ csl.ec.t.kanazawa-u.ac.jp

Specification and Bounded Model Checking for Preemptive Scheduling Systems Using Stopwatch Automata(Extended Abstract)

S. Takinai, S. Yamane
Graduate School of Natural Science,Kanazawa University
takinai @ csl.ec.t.kanazawa-u.ac.jp

In this paper, we propose formal verification for preemptive scheduling systems using stopwatch automata [1, 2]. We propose the followings:

1. First, we model preemptive scheduling systems using stopwatch automata, and then compose them with timed automaton and scheduler automaton.
2. Next, we encode them as math-formula, and then verify them by bounded model checking.

1 はじめに

組込みシステムは, 低消費電力など厳しいリソース制限を課されながら, 高い信頼性を要求されており, しばしばプリエンプティブスケジューリングシステムである. プリエンプティブスケジューリングシステムとは, CPUが優先度の低いタスクに割り当てられているときに, 優先度の高いタスクが起動した場合, 割り込みが発生し, CPUが優先度の高いタスクに横取りされるようなシステムである. [3].

本論文では, そのようなプリエンプティブスケジューリングシステムが, 要求された信頼性を有しているかを, 有界モデル検査を用いた形式的手法によって検証する方法を提案する. 本論文の信頼性の検証とは, システム全体がある性質を満たしているかどうかを判定することであり, 例えば, スケジューラビリティが満たされていることや, デッドロックが生じないこと, 排他制御が正しく行われることなどである.

リアルタイムシステムの仕様記述に関する重要な研究として, R.Alur と D.L.Dill の時間オートマトン [4] があるが, それではプリエンプティブスケジューリングシステムの仕様記述は不可能である. そこで, 我々は, R.Alur と T.A.Henzinger のハイブリッドオートマトンのサブクラスである, ストップウォッチオートマトン [1, 2] を用いたプリエンプティブスケジューリングシステムの記述方法を用いる.

ストップウォッチオートマトンの到達可能性解析問題は一般に計算不能である [5]. ストップウォッチオートマトンで初めてプリエンプションを表現した R.Alur の手法 [1] は, その停止性が保証されていない. Y.Abdeddaim らは Job-Shop という有限な動作をする非周期タスク群を対象に, 制限されたストップウォッチオートマトンでプリエンプションを起こすシステムの動作を記述して,

最適スケジューリング問題の決定可能性を示したが [2], 彼らの提案は周期タスクでは使うことができない. また K.G.Larsen らはストップウォッチオートマトンの取りうる時間領域の中で, ゾーンで表現できない部分を大きめに近似することでゾーンとして表現する方法を提案した [6].

我々が提案する検証手法は以下である. (1) リアルタイム OS と周期タスクからなるシステムを対象に, まずストップウォッチオートマトンと時間オートマトンを用いてシステムの構成要素のモデルを作成する. (2) 次に, それらを並列合成することで一つのストップウォッチオートマトンを作成し, (3) 最後に, このモデルを対象に有界モデル検査 [7, 8] を行う. この有界モデル検査は, ストップウォッチオートマトンの意味をストップウォッチ kripke 構造として解釈し, また確認したい性質を ICTL(Integrator CTL) で表現することで, 論理式の充足可能性問題として扱われる. 有界モデル検査の制限時間は, システムの一周期, つまりシステムに含まれるすべての周期タスクの周期の最小公倍数の時間とする. これにより, 検証を有限の時間で確実に停止させながら, システムの取りうる全ての動作を網羅的に検証する.

本研究ではタスクのみならず, 環境や OS の機能の一部までもモデル化する. このような, システムの広範囲に及ぶモデル化の事例は過去に報告されておらず, 本研究独自のアイディアである. また, 問題をモデル検査としてではなく SAT として扱うという点で, T.A.Henzinger らの提案したハイブリッドオートマトンのモデル検査器 Hytech[9] とアルゴリズムが異なっており, 検証の所要メモリ量の削減が期待される.

2 プリエンプティブシステムの設計検証手法

本論文で提案するプリエンプティブスケジューリングシステムの検証方法は以下の通りである.

1. まず, システムに含まれるすべてのタスクを, R.Alur らのストップウォッチオートマトン [2] で各々記述する. なお, 本論文ではこのストップウォッチオートマトンに対し, 内部動作を詳細に記述するために, その機能の一部をステートチャートで表現できるよう拡張を施している.

2. 次に, すべての周期タスクの起動タイミングを一括記述する, 環境オートマトン E(A) を時間オートマトンで記述する. この時間オートマトンは, システムに対して起動イベントを送信する外部環境をモデル化したものある.

3. さらに, プリエンプティブスケジューリングポリシィをエンコードしたスケジューラオートマトン E(Sch) を時間オートマトンで記述する.

4. 最後に, 1-3. で構築した各オートマトンを並列合成し, 新たなストップウォッチオートマトンを作る. そして, システムが満たすべき性質と, システムが取りうる動作を網羅できる期間を入力とした, 有界モデル検査を行う. Fig.1 は検証手順の概念図である.

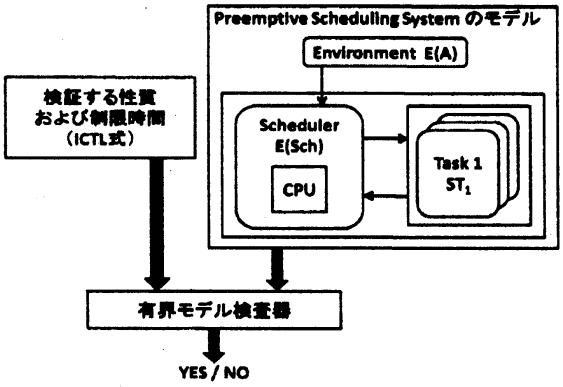

Fig. 1: 検証手順

ただし, 本論文で扱うプリエンプティブスケジューリングシステムは, 以下の前提を持っているとする.

1. タスクを実行する CPU はシングルコアである. すなわち, 同時に 1 つまでしかタスクの実行ができない.

2. タスクはすべて, 周期的に外部環境によって起動される周期タスクである. ただし, 仮にシステムに非周期タスクが含まれていても, それが周期タスクによって監視, 起動されるものであれば, それらもまた周期タスクとして扱うことができる.

3. タスクは, 最初に全て起動される. すなわち, システムはクリティカルな状態から動作を始める.

4. タスクには, それぞれ固定優先度がレートモノトニックスケジューリングによって設定さ

れている．この前提は，一般的なリアルタイムシステムにおいてレートモノトニックスケジューリングが採用されていることに因るものであり，優先度が固定されるのであれば，レートモノトニックスケジューリング以外のアルゴリズムで優先度が決定されても，本手法を適用することができる．

3 ストップウォッチオートマトンによるシステムのモデル化

プリエンプティブスケジューリングシステムのモデル化にあたり，まずシステムに含まれる周期タスクを R.Alur らのストップウォッチオートマトンで記述する．

以降のモデルに含まれる，タイミング制約式は $x_i \sim d_i$ または $x_i - x_j \sim d_{ij}$ の論理積 ϕ であり，その集合を $B(C)$ と表記する．ただし，$C = \{c_1, c_2, ...\}, \sim \in \{<, >, ==, <=, >=\}$, $x_i, x_j \in C$, d_i と d_{ij} は整数である．アクション Act は各オートマトンが同期して動作するための共通のラベルである．Act_{in} は入力イベントを意味するアクションであり，アクションの名前に?をつけて表現される．このアクションがついた遷移は同じ名前の出力イベント Act_{out}（これは!で表現される）がついた遷移と同期して行われる必要がある．アクションが τ であれば，その遷移は内部遷移である．

定義 1 (ストップウォッチオートマトン)
周期タスクを記述するストップウォッチオートマトンは ST_i=$(Q_i, q_{i0}, C_i, I_i, Act, \mathbf{u}_i, E_i)$ で表される．ここで，

1. Q_i はロケーションの有限集合，
2. q_{i0} は初期ロケーション，
3. C_i は n 個のクロック変数の集合，
4. $I_i : Q_i \rightarrow B(C_i)$ は各ロケーションに不変条件 ϕ_i を割り当てる関数，
5. Act はアクションの有限集合 $Act_{in} \cup Act_{out} \cup \{\tau\}$，
6. $\mathbf{u}_i : Q_i \times \dot{C}_i \rightarrow \{0,1\}^n$ は各ロケーションにおいて 0 または 1 の勾配（時間微分係数）をすべてのクロック変数に割り当てる関数，
7. $E_i \subseteq Q_i \times B(C_i) \times Act \times 2^{C_i} \times Q_i$ は遷移関係の集合である．

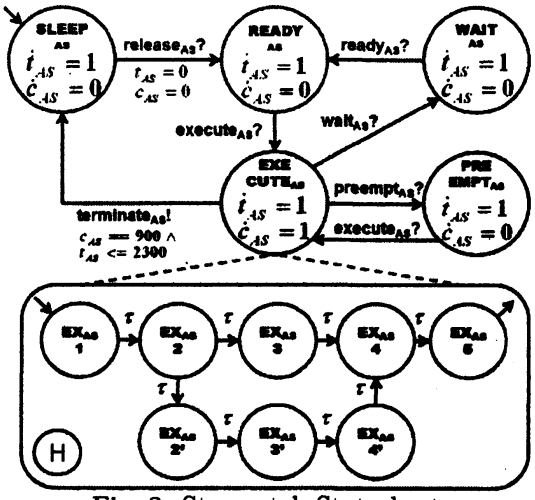

Fig. 2: Stopwatch Statechart

また，$\dot{C}_i$ はクロック変数の時間微分の集合である．遷移関係 E_i の要素は $(q_i, \phi_i, a, \lambda_i, q_i\prime) \in E_i$ であり，$q, q\prime \in Q_i$ はそれぞれ遷移元と遷移先のロケーション，$a \in Act$, $\lambda_i \subseteq C_i$ はリセットされるクロック変数の集合，ϕ_i は不変条件である．

タスクの内部動作を詳細に記述するため，文献 [11] で提案したストップウォッチオートマトンをストップウォッチステートチャートとして表現する手法を利用する．詳細な定義は省略するが，ストップウォッチステートチャートの主要な特徴としては，全体が 2 階層のオートマトンとなっており，上位階層は制限付きストップウォッチオートマトンであるが，下位階層は時間オートマトンであり，そこではヒストリーを用いるという点が挙げられる．

例 1 (ストップウォッチステートチャートの例)
Fig.2 は *K.G.Larsen* らのオーディオ／ビデオプロトコルの *Media Server* アプリケーション *[13]* のメディアサーバーシステムに含まれる周期タスク「*AS-Controller*」の内部動作をストップウォッチステートチャートで記述したものである．ここで図中の，"=="は右辺と左辺の値が等しいときに *true* となる演算子を意味し，"="は右辺の値を左辺に割り当てることを意味している．

タスク *AS-Controller* は，起動されていない状態を示すロケーション $SLEEP_{AS}$，起動されたのち *CPU* が割り当てられるのを待っている状態を示す $READY_{AS}$，実行状態である $EXECUTE_{AS}$ の順で遷移し，再び $SLEEP_{AS}$ に戻ることが，一回の処理を終えたことを意味する．実行状態においてのみ，計算時間を計測する

クロック変数 c の時間微分係数が 1 となっている. 他のタスクによって実行状態から割り込まれ状態を示す $PREEMPT_{AS}$ へ遷移すると, 計算時間は増加せず, 起動されてからの経過時間 t のみが増加する. $EXECUTE_{AS}$ では下層でさらに細かい状態が用意されており, 実行内容 (プログラム) に含まれる条件分岐やループなどの細かい動作はここに記述される. ロケーション $EX_{AS}2$ のように, 複数の内部遷移が可能なロケーションも存在するため, 検証段階では選択可能な全てのパスについて網羅的に探索を行う. 処理を終えるまでに他の状態に遷移する場合, 下層のどこに滞在していたかは履歴に保存する.

下層の i や c の値は, 上層部の値をそのまま引き継いだものであるため, 図では省略している.

ストップウォッチステートチャートは, 上層部分はすべて共通であるが, 下層部分が検証対象とするタスクの特性によって変化する.

次に, すべてのタスクを周期的に起動させる外部環境をモデル化した, 環境オートマトン **E(A)** を時間オートマトンで記述する.

定義 2 (環境オートマトン E(A))
環境オートマトンは 6 つ組 $(Q_A, q_{A0}, C_A, I_A, Act, E_A)$ である. ここで,

1. Q_A はロケーションの有限集合,
2. $q_{A0} \in Q_A$ は初期ロケーション,
3. C_A はクロック変数の有限集合,
4. $I_A : Q_A \rightarrow B(C_A)$ は各ロケーションに不変条件 ϕ_A を割り当てる関数,
5. Act はアクションの有限集合 $Act_{in} \cup Act_{out} \cup \{\tau\}$,
6. $E_A \subseteq Q_A \times B(C_A) \times Act \times 2^{C_A} \times Q_A$ は遷移関係の集合である.

ストップウォッチオートマトンと異なり, 全てのロケーションにおいて時間微分係数は 1 となっているため, クロック変数はどのロケーションにおいても単調増加して行く.

例 2 (環境オートマトン E(A) の例)
Fig.3 に環境オートマトン *E(A)* の例を示す. *Fig.3* の環境オートマトンは, 2 つのタスク *DT* および *AS* を, まず最初に二つとも起動させる. その後は, タスク *AS* を時間が *2400* 経過するごとに, そしてタスク *DT* を時間が *2000* 経過するごとに起動させるという動作を行う.

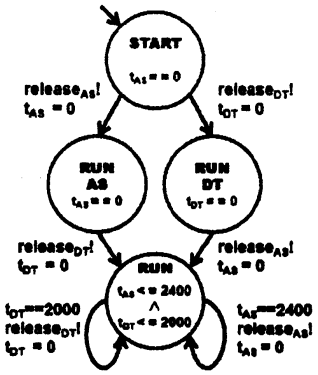

Fig. 3: Timed Automaton E(A)

どちらのタスクが最初に起動されるかは非決定であるが, どちらが先に起動されようとも, 次のロケーションからさらに次のロケーション *RUN* への遷移で, もう一方のタスクが起動される. *RUN* に至るまでの全てのロケーションに時間の経過を許さない不変条件が設定されているため, どちらのパスが選択されても, 時間的にみると 2 つのタスクは全く同時に起動することになる.

環境オートマトンは, 検証対象とするシステムに含まれるタスクの個数やそれぞれの周期などによって, その全体が変化する.

さらに, TIMES[10] のアイディアを用いて, プリエンプティブスケジューリングポリシィをエンコードした, スケジューラオートマトン **E(Sch)** を時間オートマトンで記述する. 用いられる時間オートマトンの定義は基本的に **E(A)** と同じ 6 つ組の $(Q_{Sch}, q_{Sch0}, C_{Sch}, I_{Sch}, Act, E_{Sch})$ であるが, タスクが待ち行列に存在するかどうかを, タスクの数だけのブール値で保存するタスクキュー *que* を扱う関係上, 以下の点が異なる.

1. 遷移関係のタイミング制約式に, キューの中のタスクの有無を真偽値で返す関数 Exist を含む. よって, **E(Sch)** のタイミング制約式 $B(C)$ は, タイミング制約式は $x_i \sim d_i$ または $x_i - x_j \sim d_{ij}$, または Exist(k, que) の論理積 ϕ_{Sch} となる. ただし, k はタスクの数だけ用意される, タスクを識別するための ID であり, que は参照するキューを意味する.
2. 離散遷移を行う際, キューに対してタスクの追加, あるいは削除を行う. それぞれ $que = Task_i :: que$, $que = !Task_i :: que$ と表記する. これらの, キューの更新式 $f \in F$ が追加されるため, スケジューラオートマトン

E(**Sch**) の遷移関係 E_{Sch} は $Q_A \times B(C_A) \times Act \times 2^{C_A} \times F \times Q_A$ となる.

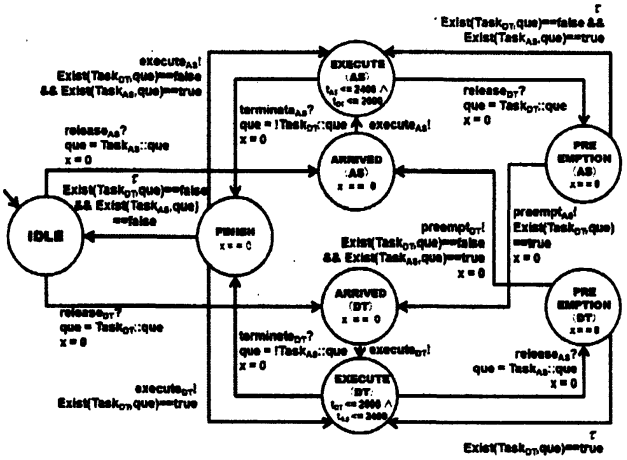

Fig. 4: Timed Automaton E(Sch)

例 3 (スケジューラオートマトン E(Sch) の例) *Fig.4* にスケジューラオートマトン *E*(**Sch**) の例を示す. *Fig.4* で示しているのは, *2* つの周期タスク (*AS* と *DT*) に対して *CPU* の割り当てを行うスケジューラのモデルである. このスケジューラは, 事前に設定された固定優先度に従い, 同時に *1* つまでのタスクを実行状態にする.

図中の $Task_i$ と $Task_j$ はタスクであり, x は状態遷移のタイミングを管理するためのクロック変数である. このオートマトンの初期ロケーションは待ちを意味する *IDLE* ロケーションである. 環境オートマトンから *release* アクションを通知されると, 該当するタスクの *ARRIVED* ロケーションへ遷移する. その後, *EXECUTE* ロケーションへ遷移すると同時に, 該当するタスクに *CPU* を割り当て, 実行状態にする. 無事処理が完了さすれば *FINISH*, *IDLE* と遷移してゆくが, もし処理途中でより優先度の高いタスクの起動が通知された場合は, 処理中であったタスクを *preempt* アクションを通知することで一時停止させ, 優先度の高いタスクに *CPU* を割り当てるという動作を行う.

このように, スケジューラオートマトンは, リアルタイム *OS* の機能の一部 (スケジューラ) を表現する.

プリエンプティブスケジューリングシステムのモデリングの最終段階として, ここまでに作成した, システムに含まれるタスクの個数分のストップウォッチステートチャート STi(i=1,2,...), ただ 1 つの, タスク起動を管理する環境オートマトン E(**A**), ただ 1 つの, タスクのスケジューリングを行うスケジューラオートマトン E(**Sch**) を**並列合成**する. 時間オートマトン E(**A**), E(**Sch**) はストップウォッチオートマトンのサブセットであるため, これらを並列合成すると新たなストップウォッチオートマトンが 1 つ生成される. その定義は以下で表される.

定義 3 (並列合成ストップウォッチオートマトン) 並列合成後のストップウォッチオートマトン $A = E(\mathbf{A}) \times E(\mathbf{Sch}) \times ST_i$ は, 7 つ組 $(Q, q_0, C, I, Act, \mathbf{u}, E)$ である. ここで,

1. Q はロケーションの有限集合であり, 合成するすべてのオートマトンのロケーションの直積 $Q_A \times Q_{Sch} \times Q_i$,
2. $q_0 \in Q$ は初期ロケーションであり, 合成するすべてのオートマトンの初期ロケーションの組 $< q_{A0}, q_{Sch0}, q_{i0} > \in Q_A \times Q_{Sch} \times Q_i$,
3. C はクロック変数の有限集合であり, 合成するすべてのオートマトンに含まれるクロックの和集合 $C_A \cup C_{Sch} \cup C_i$,
4. $I : Q \rightarrow B(C)$ は各ロケーションに不変条件 ϕ を割り当てる関数であり, $\phi \in B(C)$ は $\phi = \phi_A \wedge \phi_{Sch} \wedge \phi_i$, ただし, $\phi_A \in B(C_A), \phi_{Sch} \in B(C_{Sch}), \phi_i \in B(C_i)$,
5. Act はアクション (ただし, この並列合成は *CSS* 的なスタイル *[12]* にのっとった同期合成であるため, 対となる入力アクション Act_{in} と出力アクション Act_{out} が打ち消しあい, すべて τ のアクションとなる),
6. $E \subseteq Q \times B(C) \times Act \times 2^C \times F \times Q$ は遷移関係の集合であり, 合成するすべてのオートマトンに含まれる遷移関係のうち, アクションが Act_{in} のものと Act_{out} のものを一つにまとめたものであり, $\langle q_1, \phi_1, a!, \lambda_1, q_1' \rangle \in E_i$ と $\langle q_2, \phi_2, a?, \lambda_2, f, q_2' \rangle \in E_{Sch}$ は $\langle (q_1, q_2), \phi_1 \wedge \phi_2, \tau, \lambda_1 \cup \lambda_2, f, (q_1', q_2') \rangle \in E$ とする,
7. $\mathbf{u} : Q \times \dot{C}_i \rightarrow \{0,1\}^n$ は各ロケーションにおいてタスク ST_i のクロック変数に *0* または *1* の勾配 (時間微分係数) を割り当てる関数である.

このオートマトンの状態は, 現在のロケーション $q \in Q$ とクロック変数の集合への自然数の割

り当て $\nu : C \rightarrow \mathbf{R}$, タスクキューの状態を示す que を追加した $< q, \nu, que >$ で表され, その動作は以下の二通りである.

1. 連続遷移の集合 T_δ

   システムがある一つの処理を継続して行い, それとともに時間が経過している様を表す遷移である. その要素は $\langle q, \nu, que \rangle \xrightarrow{\delta} \langle q, \nu', que \rangle$ となる. CPU を割り当てるタスクが切り替わるときは離散遷移が生じるため, 連続遷移が続いている限りキューが変化することはない. 同じ処理を続けているためにロケーションも変化せず, クロック変数 (ただし, ロケーション q において勾配 $\dot{c}$ が 1 になっているもののみ) が実数 δ だけ増加する $(\nu' = \nu + \delta)$. この実数 δ の値は, 値が増加するクロックのすべてが不変条件 ϕ を満たす範囲の上で, 任意である.

2. 離散遷移の集合 T_d

   システムが処理を切り替える瞬間を表す遷移である. その要素は $\langle q, \nu, que \rangle \xrightarrow{\phi, \tau, \lambda, f} \langle q', \nu', que' \rangle$ となる. ここでの, ϕ, τ, λ, f は並列合成ストップウォットオートマトンの遷移関係 E の要素 $(q, \phi, \tau, \lambda, f, q')$ のそれである. ロケーションが q から q' へと変わる. クロック変数の値の増加はないが, 遷移条件でリセット対象 λ に含まれていたクロック変数は 0 へとリセットされる $(\nu' = \nu[\lambda \rightarrow 0])$. que' は, que にタスク $Task_i$ が追加されたキュー (f が $que = Task_i :: que$ のとき) か, que からタスク $Task_i$ が削除されたキュー (f が $que = !Task_i :: que$ のとき), あるいは, que と同じキュー (どちらでもない場合) である. 並列合成オートマトンではアクションが全て τ となっているため, アクションの出力や入力は生じない. 離散遷移を行うには, タイミング制約式 ϕ が満たされている必要がある.

並列合成したストップウォッチオートマトンは, この二通りの動作を繰り返す. これにより, システムの動作がモデルの動作列で表現される.

4 ストップウォッチオートマトンの有界モデル検査

最後に, 並列合成して新たに構築されたストップウォッチオートマトン $E(\mathbf{A}) \times \mathbf{E(Sch)} \times \mathbf{ST_i}$ に対して, そのシステムが満たすべき (あるいは満たすべきではない) 性質と, システムが取りうる動作を網羅できる期間を入力とした, **有界モデル検査**を行う.

有界モデル検査とは, A.Biere と E.M.Clarke らによって提案された [14], あらかじめ制限時間を設定し, 検査の途中で制限時間が来ると, そこで打ち切るという検査方法である.

有界モデル検査には, 検査に制限時間を加えることで, モデル検査でしばしば発生する, 計算がいつまで経っても収束しないという問題が回避されるという利点がある. だがその反面, 検査を打ち切ったあとのシステムの動作を保障できないという欠点がある.

以下の理由により, **システムの一周期**を制限時間とすれば, 有界モデル検査でシステムの将来にわたる性質を確認することができる. ここで言うシステムの一周期とは, システムに含まれる全てのクロック変数が同時に再び 0 となるタイミング, すなわち, システムが動作しはじめてから, すべてのタスク周期の最小公倍数が経過した時点である.

1. 2節に記述された前提条件より, タスクの起動タイミングおよび優先度は一定であるため, 条件が同じであれば, システムは同じ動作を繰り返す.

2. システムが動作を始めてから, 全ての周期の最小公倍数が経過した時点で, 全てのタスクは同時に起動する. これは, システムが動作を始めたときと同じ状況である.

3. 1-2. より, システムが動作を始めてから, そこに含まれるタスクの最小公倍数まで動作を確認すれば, それ以降の動作はそこまでの動作の繰り返しであり, 異なる結果が出てくることはない.

タスクの起動順が非決定であることや, 下層ステートチャート内に分岐が含まれることから, 初期状態からシステムの一周期が経過し, 再び初期状態に戻るまでの動作列は複数存在することになる. 本研究では, **初期状態へ至るすべてのパスを網羅的に一度で確認する**ため, パスの取りこぼしは発生しない.

我々は, ストップウォッチオートマトンに対して有界モデル検査を行うにあたり, ストップウォッチオートマトンの意味を**ストップウォッチ kripke 構造**として記述し, また検証したい性質を **ICTL**[15] で記述することで, ストップウォッチオートマト

ンに対する有界モデル検査の問題をSAT（充足可能性問題）として扱う．

定義 4 (ストップウォッチ kripke 構造)
ストップウォッチ *kripke* 構造 A' は (Q', q'_0, Δ) の3つ組からなる．

1. Q' は状態の有限集合 $\langle q, \nu \rangle$, ただし $q \in Q$, ν はクロック変数に正の実数を割り当てる関数であり $\nu : C \to R$,
2. $q'_0 \in Q'$ は初期状態 $\langle q_0, \mathbf{0} \rangle$,
3. Δ は状態間の遷移関係 $Q' \times (T_d \cup T_\delta) \times Q'$ である．

初期状態でのロケーションはストップウォッチオートマトン A の初期ロケーションであり，またクロック変数の値はすべて初期値（0）である．

遷移関係は，遷移元の状態から離散遷移 T_d または連続遷移 T_δ を行い，遷移先の状態へと移り変わるという関係を示している．アクションは並列合成の際にすべて τ となるため，ストップウォッチ *kpipke* 構造には表れない．また，ストップウォッチオートマトン A に不変条件を割り当てる関数 I はすべて遷移関係 Δ に含める．そして，各ロケーションにすべてのクロック変数に勾配を割り当てる関数 $\mathbf{u}$ や，キューへのタスクの追加・削除という操作も状態間の遷移関係 Δ に以下の形で取り込まれる．

$$T_\delta = ((z' - z > 0) \wedge (q' = q) \wedge \bigcap_{x \in X_1} (x' = x) \wedge (y' = y))$$

$$T_d = (q_i \wedge \phi \wedge q'_j \wedge \bigcap_{x \in \lambda} (x' = z') \wedge \bigcap_{x \notin \lambda} (x' = x) \wedge que' \wedge (z' = z) \wedge (y' = z') \wedge \bigcap_{c \in C(q_i)} (c' = c + z - y) \wedge \bigcap_{c \notin C(q_i)} (c' = c))$$

$'$ 付きの値は遷移後の値であることを意味している．

連続遷移 T_δ が生じる際に満たされる式とは，システムが起動してからの経過時間を計るクロック z が遷移によって増加しており $(z' - z > 0)$，かつ，ロケーションに変化はない $(q' = q)$ というものである．

離散遷移 T_d が生じる際に満たされる式とは，まず遷移前と遷移後のロケーション q_i, q'_j がともに真であり，遷移後のキュー内の各タスクの有無を示すブール値の結合 que' が真であり，また不変条件 ϕ も満たされ，かつ，システムが起動してからの経過時間 z に変化がない $(z' = z)$ というものである．離散遷移の際，リセットされるクロックの集合に含まれる $(x \in X_1)$ クロックは，式 $x' = z'$ によりその値がシステムが起動してからの経過時間 z となる．このように，クロック変数の値は，値そのものではなく，前回リセットされた時間との差という形で，遷移の条件式 ϕ において参照される．

時間微分係数によってクロックの増加の有無が変化するストップウォッチオートマトンの特性は，クロック変数 y と c で以下のように表現している．

まず，クロック変数 c は計算時間を蓄積するクロックである．離散遷移が生じるたびに，クロック変数 c には前回の遷移の間に経過した時間 $(z-y)$ が加算される．ただし，この加算は前回遷移が生じたロケーション q_i において勾配が *1* であるもの $(c \in C(q_i))$ だけを対象に行われ，勾配が *0* であったクロック $(c \notin C(q_i))$ に対しては加算は行われない．離散遷移が生じるたびに，連続遷移の開始時間を意味するクロック変数 y にシステムの起動からの経過時間 z が代入されるため $(y' = z')$，その後連続遷移が生じたときのみ計算時間 $(z - y)$ が正の値となり，計算時間をクロック変数 c に蓄積することが可能となる．

検証する性質はICTL（積分計算木論理）で記述する．ICTL は TCTL を，時間について任意の勾配を設定できるよう拡張したものであり，ストップウォッチオートマトンの (0,1) の間で勾配が変化するという性質を表すには ICTL を用いる必要がある．

検証したい性質の記述例は以下である．

$$\forall\Box(t_{AS} : true).(c_{AS} : Q_{AS} = execute_{AS}). \forall\Box(t_{AS} - 1000 \leq c_{AS})$$

この式の意味するところは，タスク AS のロケーション Q_{AS} が $execute_{AS}$ となったときに勾配が 1 となるクロック変数 c_{AS} と，常に勾配が 1 である t_{AS} について，システムがとりうるあらゆるパスで必ずその差が 1000 に達してはならないというものである．すなわち，この式が充足可能という結果が得られれば，そのシステムには，タスク AS が一度実行状態に入れば，その後 1000 単位時間以上，割り込みによって待たされることはないという性質があるということが言える．

ここまでに定義したストップウォッチ kripke 構造と，ICTL を入力とし，SAT Solvr によって充

足可能性を判定する. ストップウォッチオートマトン上での到達可能性判定は一般に計算不能であるが, 我々が提案する周期の最小公倍数までを制限時間として行う有界モデル検査の計算量は, 蓄積する経過時間が無限とならないため PSPACE 完全となる [16].

5 まとめ

本論文では, ストップウォッチオートマトンと時間オートマトンを用いた, プリエンプティブスケジューリングシステムの形式的なモデル化, 仕様記述, 検証方法を提案した. これにより, 設計段階で周期タスクからなるプリエンプティブスケジューリングシステムの信頼性を数学的に確認することが可能であることを示した. また, 有界モデル検査を SAT として実装する手法を提案し, その計算量を示すことで, この検証手法の実用性を示した. 今現在我々は, 本研究の提案手法を SMT(SAT modulo theory) を用いて実装し, 提案内容の有効性の確認を実証実験により進めているところである.

6 謝辞

本研究の発展に関わる重要なヒントを与えてくださった大阪大学の土屋准教授, 北陸先端科学技術大学の平石教授, 小川教授のお三方に深く感謝いたします.

参考文献

[1] R. Alur, T.A. Henzinger, Pei-Hsin Ho. Automatic symbolic verification of embedded systems. *RTSS*, pp. 2-11, 1993.

[2] Y.Abdeddaim, O.Maler: Preemptive Job-Shop Scheduling Using Stopwatch Automata. *LNCS 2280*, pp.113-126, 2002.

[3] G.Buttazzo. Hard Real-time Computing Systems:Predicatable Scheduling Algorithms and Applications. *Kluwer Academic Publishers*, 1997.

[4] R.Alur, D.L.Dill: Automata For Modeling Real-Time Systems.. *LNCS 443*, pp.322-335, 1990.

[5] R Alur, C. Courcoubetis, T.A. Henzinger, Pei-Hsin Ho: Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. *LNCS 736*, pp.209-229, 1992.

[6] F.Cassez, K.G.Larsen: The Impressive Power of Stopwatches. *LNCS 1877*, pp.138-152, 2000.

[7] G.Audemard, A.Cimatti, A.Kornilowicz, R.Sebastiani: Bounded Model Checking fot Timed Systems. *LNCS 2529*, pp.243-259, 2002.

[8] G.Audemard, et al.: Verifying Industrial Hybrid Systems with MathSAT. *LNCS 119*, pp.17-32, 2004.

[9] T.A.Henzinger, P.-H.Ho, and H.Wong-Toi. HyTech: A Model Checker for Hybrid Systems. *Software Tools for Technology Transfer 1*, pp.110-122, 1997.

[10] T.Amnell, et al.: Times - A Tool for Modelling and Implementation of Embedded Systems. *LNCS 2280*, pp.460-464, 2002.

[11] S.Yamane, S.Takinai: 組込みシステムの UML 分析設計からタスク設計までの設計検証方法論. 情報処理学会研究報告, 2007-SE-157,pp.87-94, 2007.

[12] R.Milner: communicating and mobile systems:the π-calculus *CAMBRIDGE UNIVERSITY PRESS 1999.*

[13] K.Havelund, et al.: Formal modeling and analysis of an audio/video protocol: an industrial case study using UPPAAL. *RTSS*, pp.2-13, 1997.

[14] A.Biere, et al.: Symbolic Model Checking without BDDs. *LNCS 1579*, pp.193-207, 1999.

[15] R.Alur, T.A.Henzinger, Pei-Hsin Ho Automatic Symbolic Verification of Embedded Systems. *IEEE Trans. SE*, pp.181-201, 1996.

[16] R.Alur, C.Courcoubetis, T.A.Henzinger: Computing accumulated delays in real-time systems. *LNCS 697*, pp.181-193, 1993.